Student Name: **Student id:** **Sect:** **#:**

QUESTION #	1	2	3	4	TOTAL
MAX POINTS	17	17	15	14	
POINTS EARNED					

University of Bahrain College of Information Technology

Department of Computer Science

ITCS 332: Concepts of Programming Languages FIRST TEST Date: OCT 27, 2013

****

QUESTION ONE: [17 pts]

1) Convert the following BNF rules into equivalent EBNF rules

```
<cas>    → <id> <op> <int> | <id> <op> <sign> <int>
<op>     → += | -= | *= | /= | %=
<id>     → <Let> | <id> <other>
<other>  → <Letter> | <digit>
<int>    → <digit>| <int> <digit>
<sign>   → + | -
<anyV>   → <id> | <id> ?? <anyV>| <id> ** <anyV>
```

<cas> → <id> (+=|-=|*=|/=|%=) [(+|-)] <int>

<int> → <digit> { <digit>}

<id> → <Letter> { <Letter> | <digit> }

<anyV> → <id> { (?? | **) <id> }

<cas>→ <Let>{<Let>|<dig>} (+=|-=|*=|/=|%=)
[(+|-)] <dig>{<dig>}

2) Convert the following EBNF rules into equivalent BNF rules

```
<SEQSTR>  → <STRING> # { <STRING> # }
<STRING>  → $ { <CHAR> } $
<anyN>    → [ (+ | - ) ] <N>
<CHAR>    → a|b|c ...|A|B|C| ... 0|1|2|3 ... |+|*|; ...
```

<SEQSTR> → <STRING> # | <STRING> # <SEQSTR>

<STRING> → $$ | $ <CHARS> $

<CHARS> → <CHAR> | <CHAR> <CHARS>

<anyN> → <N> | + <N> | - <N>

<CHAR> → a|b|c|...|A|B|C|...0|1|2|3|...|+|*|; ...

QUESTION TWO: Carefully study the following grammar and answer the next 3 questions. **[17 pts]**

```
<cas>     → <id> <op> <int> | <id> <op> <sign> <int>
<op>      → += | -= | *= | /= | %=
<id>      → <Let> | <id> <other>
<other>   → <Letter> | <digit>
<int>     → <digit>| <int> <digit>
<sign>    → + | -
<Letter>  → a|b|c ...z|A|B|C| ... Z
<digit>   → 0|1|2|3 ...|9
```

1) Using the above BNF rules, construct the left-most derivations for the sentence: **A4C *= -69**

2) Construct the syntax graph for each of the following nonterminals: (**<cas>**,**<sign>**), (**<id>**,**<other>**), (**<int>**) in the above-mentioned grammar.

3) Using the above BNF rules, construct the parse tree for the sentence: **A4C *= -69**.

QUESTION THREE: **[15 pts]**

1) Construct the EBNF rules to define a function heading <heading> consisting of a type <type> followed by a name <id> followed by a parameters list <parList> enclosed in parenthesis. The parameter list consists of zero or more parameters separated by semicolon. A parameter is a name followed by a colon followed by a type.

<heading> → <type> <id> ([<parList>])

<parList> → <id> : <type> { ; <id> : <type> }

2) Construct the BNF rules to define a list assignment statement <**Listass**> consisting of two sides separated by "**=**". The left side consists of one or more identifiers <**id**> separated by commas "**,**" enclosed between braces {}. The right side consists of one or more integers <**int**> separated by semicolons "**;**" enclosed between brackets [ ].

<Listass> → { <Left> } **=** [<right>]

<Left> → <id> | <Left> , <id>

<right> → <int> | <right> ; <id>

3) Write ONE EBNF rule that defines a list assignment statement <**Listass**> consisting of two sides separated by "**=:**". The left side consists of one or more identifiers <**id**> separated by hashes "**#**" . The right side consists of one or more integers <**int**> separated by two dollar signs "**$$**".

<Listass> → <id> { # <id> } =: <int> { $$ <int> }

QUESTION *FOUR*: *Fill in blanks Questions* **[14 pts]**

1) The compiler scanner accepts as input **a string of characters** and produces as output **a set of lexemes**.

2) The compiler parser accepts as input **a set of lexemes** and produces as output **a parse tree**.

3) The kind of semantics that specifies the detailed ordered steps of a problem solution is called **procedural**; while the semantics that describes the form of the results of a problem is called **declarative**.

4) The ONLY two operators allowed in a clausal form are: **CONJUNCTION** and **DISJUNCTION**

5) The body of the horn clause (rule) is called **ANTICEDENT** and the head is called **CONSEQUENT**.

6) The reliability of Java programs is improved by **Index range checking** and **Type checking.**

7) The advantages of using preprocessor macros are: **Programs with macros are more generic (flexible)** and **Programs with macros execute faster.**

8) The number of tokens in a C++ statement: **if(dt+23*ct)/17 >9)dt=st;** is **11** and the number of lexemes is **17**.

9) The bottleneck in interpreting systems is **STATEMENT DECODING**. The main bottleneck in von Neumann computers is **CPU-MEMORY SPEED GAP**.

10) Language design is influenced by 2 factors: **COMPUTER ARCHITECTURE** and **PROGRAMMING METHODOLOGIES**.

11) An attribute grammar is a context-free grammar (BNF) plus 3 additions, name any two of them: **SEMANTIC FUNCTIONS, PREDICATE FUNCTIONS, ATTRIBUTE VALUES.**

12) An example of conflicts between execution speed and reliability **Index range checking**. The language with fewer exceptions is more **Orthogonal** than one with more exceptions.

13) The predicate function of the syntax rule <expr> ::= <var> is

<expr>.actual_type == <expr>.expected_type.

14) The semantic function of the syntax rule <expr> ::= <var> is:

<expr>.actual_type ← <var>.actual_type.

{ axiomatic semantics, synthesized, semantic functions, query, type checking, lexeme, associativity rules, token, denotational semantics, sentence, derivation, sentential form, inherited, predicate functions, inference rule, unification, instantiation, operational semantics, statement decoding, precedence rules, static semantics, exception handling, parser, scanner, conjunction, disjunction, anticedent, consequent, unification , backtracking, axiom, synthesized, declarative, procedural }.